Plugin JUnit

Mikaël Marche

26 avril 2006

Junit est une API Java permettant de décrire des tests unitaires d'une application. Le plugin Junit présent dans Salomé-TMF permet d'exécuter automatiquement des tests stéréotypés « TestCase » de Junit.

Table des matières

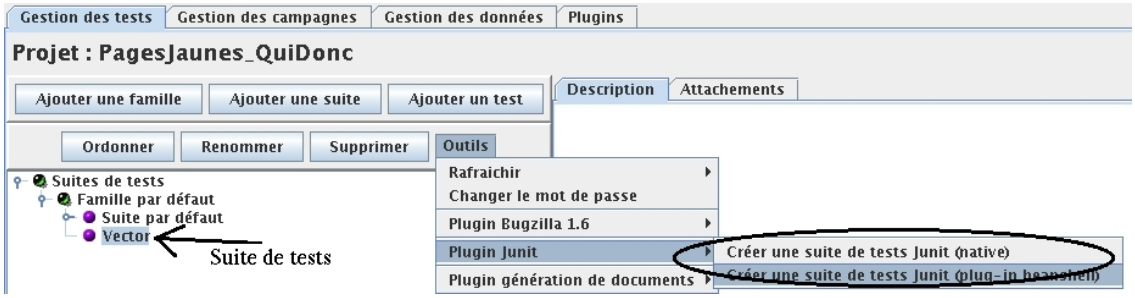
1 Créer une suite de tests Junit

Le plugin Junit permet créer une suite de tests Junit à partir des tests déclarés dans un objet stéréotypé « TestCase ». Il faut disposer pour cela d'un fichier « .jar » ou « .zip » contenant le code compilé de cet objet.

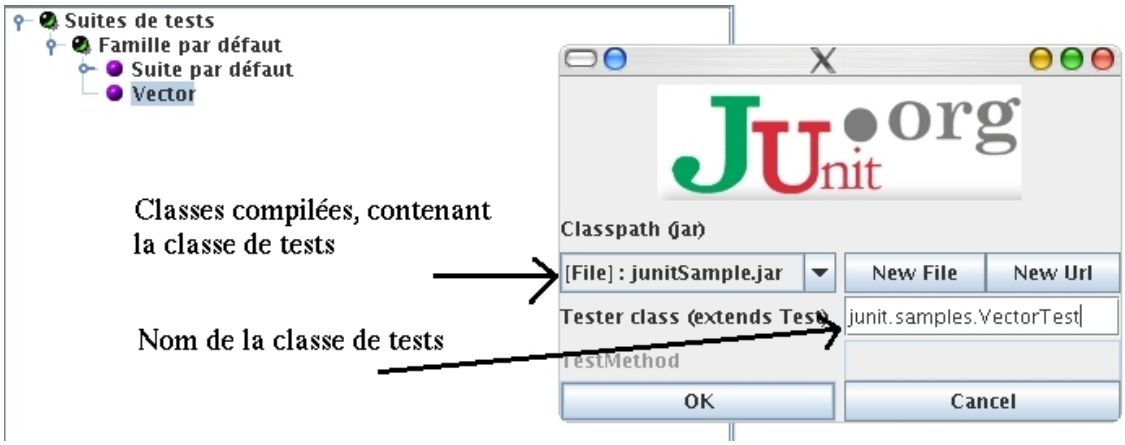
Il existe deux modes de création de suite de tests Junit, un mode dit « natif » et un autre lié au plugin BeanShell. Le mode natif utilise la réflexion Java pour instancier les tests, alors que le mode BeanShell produit des scripts de tests BeanShell qui décrivent l'instanciation et l'exécution de la classes de tests.

Dans le mode natif le fichier « .jar ou .zip » doit contenir en plus de l'objet de test stéréotypé « TestCase », l'ensemble des classes nécessaires à l'exécution des tests. Cette restriction est due au principe d'exécution du mode natif qui ne peut être lié à un script d'environnement ou d'exécution pour charger d'autres classes.

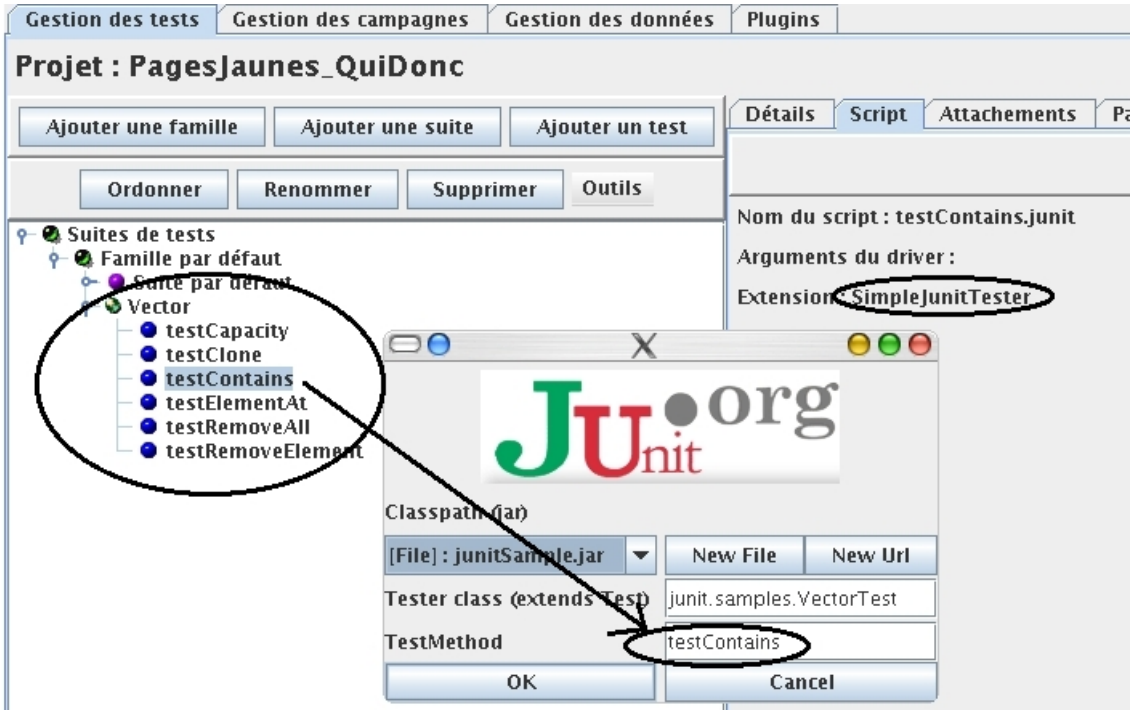
Pour créer une suite de tests Junit, après avoir sélectionné une suite de tests dans le projet, actionner dans le menu « Outils->Plugin junit » la fonctionnalité « Créer une suite de tests Junit ».



Remplissez ensuite le formulaire de création de classes comme suivant :



A partir de ces informations le plugin Junit analyse la classe de test sélectionnée pour produire l'ensemble des tests contenus sous forme de tests automatique dans Salomé (BeanShell ou natif, suivant le mode sélectionné).



2 Modifier un test Junit

La modification d'un test Junit dépend du mode de création de la suite de tests. En mode natif, le formulaire de création permet de modifier les informations « classpath, Tester class, et TestMethod » qui identifient le test. En mode BeanShell, il est possible de modifier le programme d'exécution de la classe de test (se reporter pour cela aux informations relatives au plugin BeanShell).

3 Utilisation d'objets et de paramètres de tests

Quelque soit le mode de création d'un test Junit, il est possible d'utiliser dans la classe de test Junit, les paramètres de tests liés à une exécution et les objets de l'IHM de Salomé (*=org.objectweb.salome_tmf.data).

Nom	Classe	Description
date	*Java.lang.Date*	Date de l'exécution
time	*Java.lang.Time*	Heure de l'exécution
salome_projectName	*Java.lang.String*	Nom du projet courant
salome_projectObject	**.Project*	Référence de l'objet projet de Salomé
salome_debug	*boolean*	Faut lors de l'évaluation du script dans l'éditeur
salome_ExecResultObject	**.ExecutionResult*	Référence sus les résultats d'exécution courant
salome_ExecTestResultObject	**.ExecutionTestResult*	Référence sur le résultat d'exécution du test courant
salome_CampagneName	*Java.lang.String*	Nom de la campagne courante
salome_CampagneObject	**.Campaign*	Référence de la campagne courante
salome_environmentName	*Java.lang.String*	Nom de l'environnement courant
salome_environmentObject	**.Environment*	Référence sur l'environnement courant
salome_ExecName	*Java.lang.String*	Nom de l'éxécution courante
salome_ExecObject	**.Execution*	Référence sur l'exécution courante
salome_TestName	*Java.lang.String*	Nom du test courant
salome_TestObject	**.Test*	Référence sur le test courant
salome_SuiteTestName	*Java.lang.String*	Nom de la suite de tests courante
salome_SuiteTestObject	**.TestList*	Référence sur la suite de tests courante
salome_FamilyName	*Java.lang.String*	Nom de la famille courante
salome_FamilyObject	**.Family*	Référence sur la famille courante
testLog	*Java.lang.String*	Log de tests à ajouter comme attachement à l'éxécution

En mode Junit natif, si la classe de tests possède un constructeur qui prend comme argument une Hashtable, la classe de tests est instanciée avec ce constructeur et un objet Hashtable qui contient les paramètres de tests valués ainsi que les objets précédents.

```
import junit.framework.TestCase;
import org.objectweb.salome_tmf.data.Environment;
import java.util.Hashtable;

public class  TestParam extends TestCase{

String url_serveur_Env;
String Adresse_services_schema_Env;
String url_serveur_DS;
String Adresse_services_schema_DS;

    public TestParam(String name) {
                System.out.println("Constructor with String used, arg=" + name);
    }

    public TestParam(Hashtable params) {
                System.out.println("Constructor with Hashtable used, arg=" + params);
                /*sample to salome_tmf object in Java
                Environment env = (Environment)params.get("salome_environmentObject");

                /* sample to use environment parameters */
                url_serveur_Env = env.getParameterValue("url_serveur");
                Adresse_services_schema_Env =  env.getParameterValue("Adresse_services_schema");

                /* sample to use dataset parameters */
                url_serveur_DS = (String) params.get("url_serveur");
                Adresse_services_schema_DS = (String) params.get("Adresse_services_schema");
    }

    public void testParam() throws Exception {
                System.out.println("(env) url_serveur = " + url_serveur_Env);
```

```
                System.out.println("(env) Adresse_services_schema = " + Adresse_services_schema_Env);

                System.out.println("(DataSet) url_serveur = " + url_serveur_DS);
                System.out.println("(DataSet) Adresse_services_schema = " + Adresse_services_schema_DS);
    }
}
```

En mode BeanShell, l'affectation de la Hashtable se fait à partir d'un appel de méthode « setParam(Hashtable) » si celle-ci existe dans la classe de tests.

```
import junit.framework.TestCase;
import org.objectweb.salome_tmf.data.Environment;
import java.util.Hashtable;

public class  TestParam extends TestCase{

String url_serveur_Env;
String Adresse_services_schema_Env;
String url_serveur_DS;
String Adresse_services_schema_DS;

    public void setParam(Hashtable params) {
                /*sample to salome_tmf object in Java
                Environment env = (Environment)params.get("salome_environmentObject");

                /* sample to use environment parameters */
                url_serveur_Env = env.getParameterValue("url_serveur");
                Adresse_services_schema_Env =  env.getParameterValue("Adresse_services_schema");

                /* sample to use dataset parameters */
                url_serveur_DS = (String) params.get("url_serveur");
                Adresse_services_schema_DS = (String) params.get("Adresse_services_schema");
    }

    public void testParam() throws Exception {
                System.out.println("(env) url_serveur = " + url_serveur_Env);
                System.out.println("(env) Adresse_services_schema = " + Adresse_services_schema_Env);

                System.out.println("(DataSet) url_serveur = " + url_serveur_DS);
                System.out.println("(DataSet) Adresse_services_schema = " + Adresse_services_schema_DS);
    }
}
```

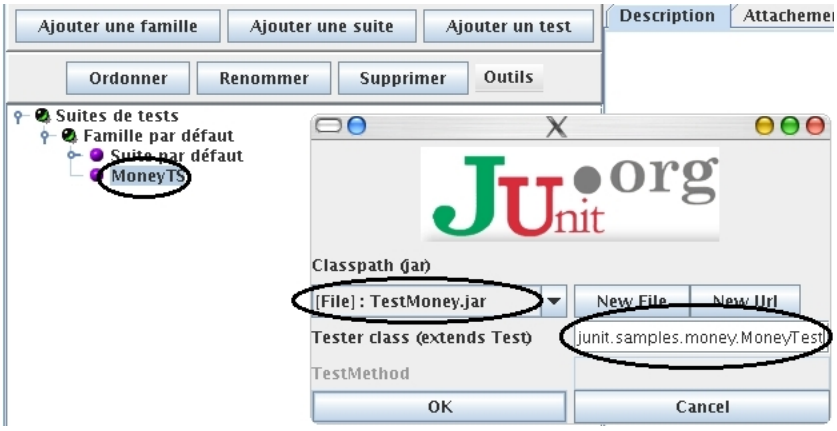
4 Exemple

L'exemple suivant présente l'utilisation du plugin Junit en mode natif et Beanshell en utilisant une classe de tests issue de la distribution Junit3.8.1. On considère en pré-requis que l'on dispose de deux fichiers « .jar », l'un contenant le code compilé de la classe ń junit.samples.money.MoneyTest ż, et l'autre contenant le code compilé de la classe sous tests « junit.samples.money.Money ».

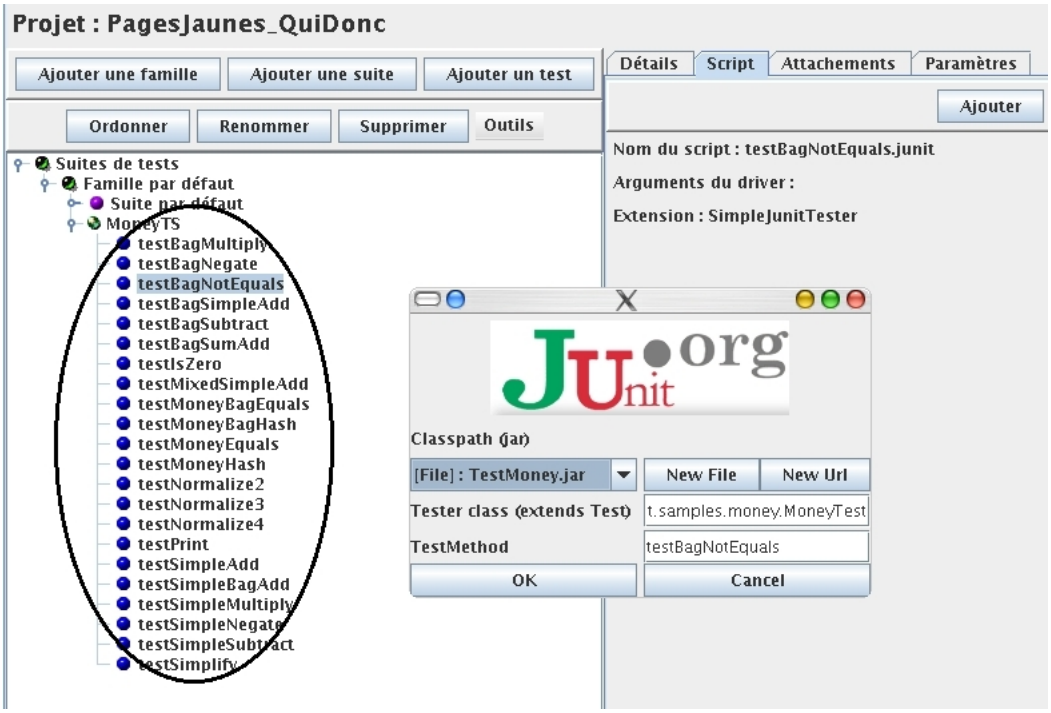
Note : L'exemple utilise les scripts d'environnement pour charger les classes de l'objet sous test, il est possible de ne pas utiliser les scripts d'environnement, si le fichier « .jar » contenant les tests contient aussi les classes de l'objet sous tests.

4.1 Création de la suite de tests Junit

Soit le fichier ń TestMoney.jar ż contenant la classe de tests « junit.samples.money.MoneyTest », on crée une suite de tests junit comme suivant :



L'ensemble des tests trouvés dans cette classe est :

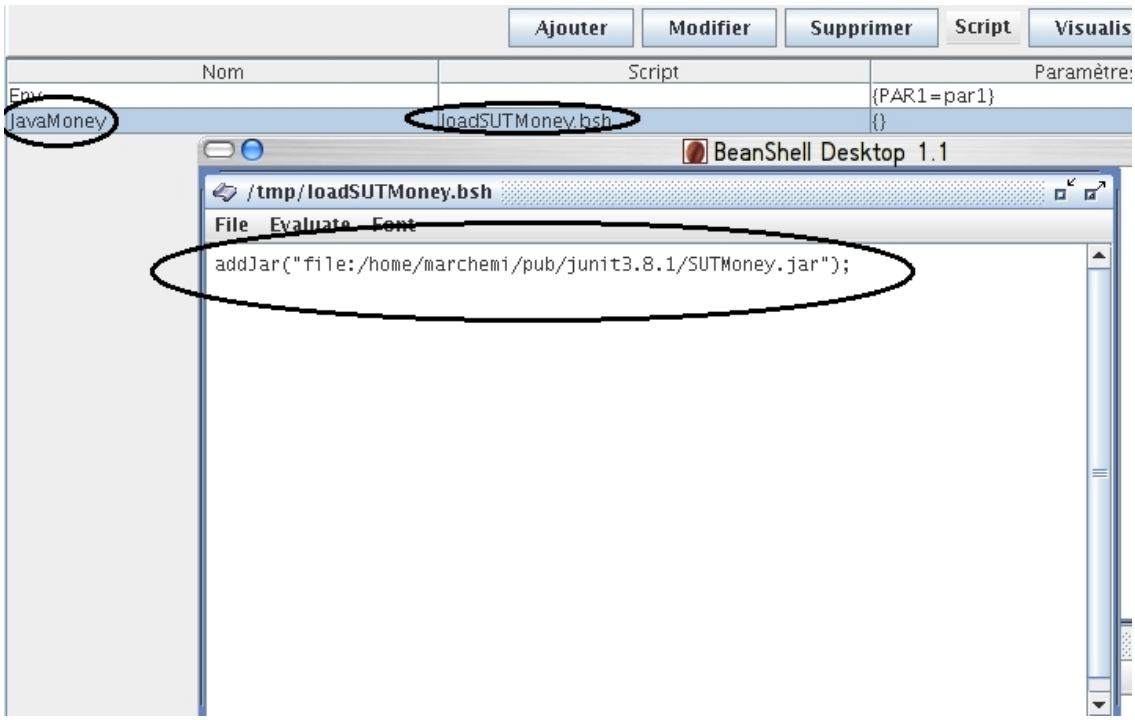


4.2 Création de l'environnement sous tests

Soit le fichier « SUTMoney.jar » contenant les classes nécessaires pour l'exécution des tests de la classe « junit.samples.money.MoneyTest ». Il s'agit maintenant de créer un environnement dont le script chargera les classes contenues dans le fichier « SUTMoney.jar ». On note que cette opération est uniquement nécessaire si le fichier « TestMoney.jar » ne contient pas toutes les classes nécessaires à l'exécution des tests.

On suppose que l'on a crée un environnement « JavaMoney », et on va ajouter à cet environnement un script BeanShell qui chargera les classes du fichier « SUTMoney.jar ». Pour cela, le script d'environnement est le suivant :

```
addJar("file:/home/marchemi/pub/junit3.8.1/SUTMoney.jar");
```



On note que le script d'environnement peut charger plusieurs fichiers « .jar », et au besoin faire des traitements en BeanShell.

On note qu'il est aussi possible de modifier le fichier *plugin.xml* du répertoire simpleJunit pour charger un ensemble de librairies Java.

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.2" "http://jpf.sourceforge.net/plugin_0_2.dtd">
<plugin id="simpleJunit" version="1" class="salomeTMF_plug.simpleJunit.SimpleJunitPlugin">
        <requires>
                <import plugin-id="core"/>
        </requires>
        <runtime>
                <library id="simpleJunit" path="simpleJunit/simpleJunit.jar" type="code"/>
                <library id="junit" path="simpleJunit/libs/HTTPUnits/junit.jar" type="code"/>
                <library id="httpjunit" path="simpleJunit/libs/HTTPUnits/httpunit.jar" type="code"/>
                <library id="xsdlib" path="simpleJunit/libs/libeXist/xsdlib.jar" type="code"/>
        </runtime>
```

```
        <extension plugin-id="core" point-id="TestDriver" id="simpleJunit.TestDriver">
                <parameter id="class" value="salomeTMF_plug.simpleJunit.SimpleJunitPlugin"/>
                <parameter id="name" value="Junit"/>
                <parameter id="description" value="Plugin Junit pour la definition de classe de tests"/>
                <parameter id="extensions" value=".junit"/>
        </extension>
        <extension plugin-id="core" point-id="Common" id="simpleJunit.Common">
                <parameter id="class" value="salomeTMF_plug.simpleJunit.SimpleJunitPlugin"/>
                <parameter id="name" value="JunitTestSuite"/>
                <parameter id="description" value="Creation de suite HTTPJunit TVMoblie"/>
        </extension>
</plugin>
```

4.3 Lancement de l'exécution

Pour exécuter la suite de tests, il suffit de créer une campagne comportant les tests Junit, et de lui associer une exécution avec l'environnement « JavaMoney ».